Building Scalable Cloud Storage

Alex Kesselman
alx@google.com
Agenda

- Desired System Characteristics
- Scalability Challenges
- Google Cloud Storage
What does a customer want from a cloud service?

- Reliability
- Durability
- Speed
- Low Cost
- Simplicity
- Big Data
- Global Access
- Atomic Updates
- Strong Consistency
Our job: Keep service complexity in the Cloud...

...and out of the Client

Providing reliability and speed isn't free; it requires a fair amount of service complexity.

Our design task is to hide most of the complexity behind the API.

We've done our job when:

1. No matter where you are or how many of you there are, our service is virtually indistinguishable from talking to a single, fast, nearby machine that provides reliable service with ~100% uptime and virtually infinite storage.

2. You don't have to write a tome of boilerplate to use the system.
Various failure modes:

What if I connect from the other side of the world?
What if the box fails?
What if the network fails?
What if data gets corrupted?
What if we need more storage?

For latency, you are forced to have boxes spread around globally. This implies having to intelligently replicate the data.

For availability, there has to be a route when boxes or networks fail, and for continued availability, those boxes must self-heal afterwards.

For data integrity, you should identify and recover from data corruptions.

For "infinite" storage and "infinite" QPS, you have to have capacity planning that has a lead time on storage use and growth.
For **durability**, error correction codes and replication.

Someone gets to monitor the service health, and try to stave off incidents that might cause unavailability.

Now, what model to use?

1. **BASE** (Basically Available, Soft state, Eventually consistent)?
   - Scales well, due to cheaper reads and writes.
   - More work for the client developer to handle resource contention.

2. **ACID** (Atomicity, Consistency, Isolation, Durability)?
   - Strong consistency means less work for client developer, but
   - Doesn't scale well, sacrificing
     - write latency
     - max QpS
Scalability Challenges

- The corpus size is growing exponentially
- Systems require a major redesign every ~5 years
- Complexity is inherent in dynamically scalable architectures
The CAP theorem states that any networked shared-data system can have at most two of three desirable properties:

1. consistency (C)
2. high availability (A)
3. tolerance to network partitions (P)

Mitigation (C & P):
- Regional architecture minimizing dependency on the global control plane
- Degraded mode of operation during partition followed by recovery
Region Can Operate in Isolation
Static sharding of object keyspace helps with hot spots, but affects listing efficiency => dynamic sharding
Maintenance overhead (e.g. billing) is proportional to the corpus size => **incremental** processing proportional to the daily churn rate

- Cold Corpus
- Deleted
- Created
It’s hardly feasible to track millions of individual users in real time => focus on top $K$ users and throttle them
Global data centers, each with Internet access, all connected through a private backbone.

Devs and Site Reliability Engineers monitor and plan.

Split the problem into two parts:

1. Data service (i.e., how to store data on the boxes)
   - Replication and sharding
   - Uses Colossus to store data (successor of GFS)

2. Metadata service (map of object names to data service entity refs)
   - Gives fairly-low latency, scalability, and limited ACID semantics
   - Uses Spanner (successor of Megastore)
End-user latency really matters
Application complexity is less if close to its data
Countries have legal restrictions on locating data
Plan and optimize data moves
Reduce costs by:
  ○ De-duplicating data chunks
  ○ Adjusting replication for cold data
  ○ Migrating data to cheaper storage
  ○ Caching reads
Continuous Multi-Criteria Optimization Problem
Replication Architecture

Fast response time; work prioritization; maximize utilization; minimize wastage.
The basics:
- Planet-scale structured storage
- Next generation of Bigtable stack
- Provides a single, location-agnostic namespace

Manual and access-based data placement:
- Distributed cross-group transactions
- Synchronous replication groups (Paxos)
- Automatic failover of client requests
• **Universe** is a Spanner deployment
• **Zone** is Unit of physical isolation
• One zonemaster, thousands of spanservers
Metadata Replication

- Each spanserver responsible for multiple tablet instances
- Tablet maintains the following mapping:
  - (key: string, timestamp:int64) -> string
- Data and logs stored on Colossus
Conclusion: Useful Principles of System Design

- Scale in multiple orders of magnitude
- Adapt dynamically to traffic changes
- Degrade gracefully in face of failures
- Support data aggregation & incremental processing
- Keep complexity at bay
More information: http://cloud.google.com/storage

